연습문제1. 함수와 제어문

문제 1.

#include <stdio.h>

int ski(int day)

{

float sum=0;

int money = 37500;

if(day<3)

{

return sum = money;

}

else if(3<=day<8)

{

return sum = money\*0.8;

}

}

int main(void){

char day=0;

while(day<7)

{

++day;

final\_money= ski(day)+final\_money;

}

printf("7일 이용 대금은 %d입니다.\n",final\_money)

}

문제 3.

#include <stdio.h>

int dr(int num,int sum){

if(num%3==0)

{

sum=sum+num;

}

return sum;

}

int main(void){

int num = 1;

int sum = 0;

while(num<1000)

{

num++

sum= dr(num,sum);

}

}

문제 4.

#include <stdio.h>

int div(int num,int sum){

if((num%4==1) && (num%6=1))

{

sum=sum+num;

}

return sum;

}

int main(void){

int num = 1;

int sum = 0;

while(num<1000)

{

num++

sum= div(num,sum);

}

}

문제 5.

#include <stdio.h>

int dr(int data[])

{

int count =0;

int i;

int n= sizeof(data)/sizeof(data[0]); // sizeof(data)= 4byte(int)\*6=28 / 4(int)=7

int result=data[0];

for(i=0;i<n-1;i++)

{

if(result<data[i+1])

result = data[i+1];

else

result = result;

}

for(j=0;j<n-1<j++)

{

if(data[j]==result)

return j;

}

}

int main

{

printf("마지막항은 %d 입니다.",dr(data[]= {7,112,28,49,14,70,28}))

}

문제 7. 레지스터는 cpu내에 있는 저장공간임.

스택은 데이터를 밀어넣는다. 나중에 넣은 자료가 먼저 나온다. push pop. 메모리를 이용할때 esp ebp를 이용한다. esp에는 스택 프레임의 끝 지점주소가 저장되어

esp는 항상 메모리의 맨위의 주소이다(pop이라는 명령어쓰면 esp의 데이터를 꺼내와서 어딘가에 넣어준다라는 뜻),32비트에서 값이 4바이트씩 변한다. ebp(extended base pointer)에는 스택 프레임의 시작 지점 주소 가장 밑부분이 저장된다. ebp 레지는 현재 사용되는 스택 프레임이 소멸되지 않는 이상 ebp 레지스터의 값은 변하지 않는다.

우선 main부터 실행이 되고, 함수가 실행된다. 스택을 통해 레지스터에 차곡차곡 넣고 함수가 끝나면 스택 맨위부터 비워, 밑에있는 main을 꺼내오는 방식이다. 함수를 여러개 써도 똑같이 작동한다.

eip레지에는 다음에 실행하여야 할 명령어가 존재하는 메모리 주소가 저장된다.

문제 10.

#include <stdio.h>

int gugudan(char i,char j)

{

return i\*j;

}

int main(void)

{

int i,j;

for (i=2;i<=9;i++)

{

for(j=1;j<=9;j++)

{

printf("%d \* %d=%d\n",i,j,gugudan(i,j));

}

}

}

문제 12. visual studio에서 디버그하는 방법

디버깅 (Debugging) 이란, 프로그램을 따라가보며 변수값의 변동을 모니터링 하는 등 프로그래밍을 하면서 발생하는 오류를 잡아내는 것이라 할 수 있다. 비주얼 스튜디오는 통합 개발 환경이라고 해서, 코드를 입력, 수정할 수 있는 에디터 (Editor), 프로그램을 실행파일로 만들 수 있는 컴파일러(Compiler)와 함께 디버거가 내장되어 있기 때문에 별도의 프로그램을 설치할 필요는 없다.

즉 코드 만들고, F5 디버깅 시작 가능, 중단점은 F9 로 설정 가능하다. 중단점을 이용해서 자신이 만든 프로그램이 얘기치 않은 동작을 할 경우, 의심가는 부분에 중단점을 설정하여 디버깅을 시작하고, 한줄씩 실행하면서 원하는대로 변수에 값이 있는지 확인하거나, 원하는대로 프로그램이 실행되는지 확인하면서 프로그램 오류를 잡아낼 수 있습니다.

그리고, 조사식 창을 이용하면 소스 코드를 고치지 않아도 변수의 값이나 메모리 주소를 바로 확인할 수 있다. 레지스터 창을 사용하면 단순히 레지스터의 내용을 보는 것 이상을 수행할 수 있다. 코드의 중단 모드에 있으면 레지스터의 내용을 클릭하여 값을 편집할 수 있다. 메모리 창을 통해 응용 프로그램에서 사용되는 메모리 공간을 볼 수 있습니다.

디버깅 정리

:1> 프로그램 오류가 발생된 곳이라고 의심되는 부분에 중단점을 설정한다.

2> '디버깅 시작'을 누른다.

3> 프로그램이 실행되다가 중단점을 만나면 그부분에서 멈춘다.

4> 'F10'으로 한줄씩 실행하면서 아래 '자동'창의 변수값을 확인하거나 프로그램이 실행된 화면을 보면서 오류를 찾는다.

5> 'Shift + F5'로 디버깅을 종료하고 오류를 수정한다.